Process

Process

- Programs have algorithms that embody a successful solution to a problem
- The problem of software development needs an algorithm or *process* that team members can follow
- Emphasis on "agile" (or responsive) processes - they incorporate a lot of feedback
- Very common examples are
  - Extreme Programming (XP)
  - Rational Unified Process

The problem

- In the past projects would last for many years before any code was delivered.
- Typically the code wasn't as requested, would be buggy, or sometimes not working at all.
- Businesses change over such a long period. The original requirements were sometimes no longer valid!
- Developers would have no idea if they were producing what the customer wanted.

Predictability

- Software development is unpredictable
  - Requirements may not be finalised.
  - Hard to predict how much effort to create software
- Reduce how far you have to predict!
  - Attempt only small chunks of work at a time.
  - You don't have to predict as far into the future.
- The feedback you get after each chunk of work is finished is a "reality check" - you have to face the reality of what you have produced, and what the client thinks of it.

Rational Unified Process

- In the 1990s three of the more commonly used software development processes were brought together to form the Rational Unified Process
- This was managed/created by the Rational Software Corporation.
- The modeling language used was UML (which they also created)
- Rational have since been bought by IBM.

Rational Unified Process

- RUP is…
  - Use Case driven
    - Use cases - descriptions of what the user wants from a system - are what the workflows are based on.
  - Architecture centered
    - The overall architecture of the system is based on the key use cases.
    - Subsections are specified, more use cases are analysed
    - This continues until a stable architecture is created.

Rational Unified Process

- RUP is…
  - Iterative and incremental
    - Rather than developing all of the requirements, then all of the analysis etc (as in the waterfall model) the RUP favours the development of complete slices of the main project.
    - Each slice is called an increment, and is the realisation of one or more use cases.
    - The benefit is that you get a lot more feedback, and earlier on, about whether your work is correct/what the user wanted.
    - You can also get paid for each successful increment, so it reduces the financial risk for both the developer and the client.

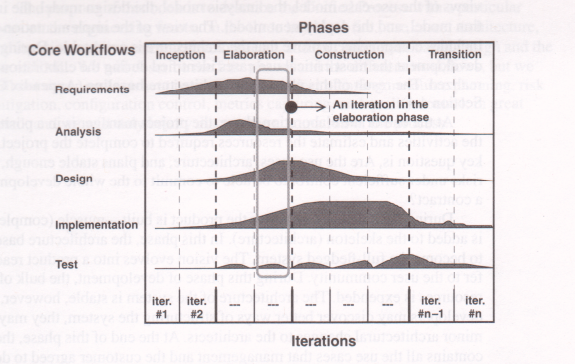
Rational Unified Process

- A project scope is performed
- Projects are divided into mini-projects
- Each mini-project is an iteration of the process, and results in an increment (a product)
- Each iteration deals with some use cases, and takes them throughout the development cycle from requirements to implementation.
- Successive increments may make preceding increments more sophisticated, or may add new use cases.

Life of the RUP

- The RUP repeats over a series of cycles making up the life of the system. Each cycle ends with a release to customers
- Each cycle consists of 4 phases…
  - Inception
  - Elaboration
  - Construction
  - Transition
- Each phase is subdivided into iterations.
- See extra document...

Overview of a cycle



The phases

- Inception
  - An idea for a product is considered in greater detail
  - A business case is created
- Elaboration
  - Use cases are specified
  - An architecture is defined.
  - At the end the project manager should be able to plan activities and estimate the resources required to build the system.

The phases

- Construction
  - The product is built
  - End milestone "does it meet user's needs sufficiently for some users to take delivery?"
- Transition
  - Product moves into beta release
  - Involves
    - Manufacturing
    - Training
    - Correction of faults

Problem

1. A programmer sets up her own software company which will specialise in her area of expertise (geographical information services).
2. Before setting up the company she researched the size of the market for her new product idea, which provides restaurant information over wireless internet to hand held devices. She also researched the possible money she could make if she co-branded her service with specific restaurants.
3. She develops a demonstration program which she shows to a small group of restaurants for feedback.
4. Based on favourable feedback she decides to go ahead with the project. She further develops the program and releases it to a single restaurant. During the first fortnight she corrects errors in the product, and changes some aspects of how the users interact with the system.
5. When she finally feels confident enough with the software, she releases the product for sale.

- Describe how the activities listed above can be presented using the Rational Unified Process in terms of phases and increment

Phases…

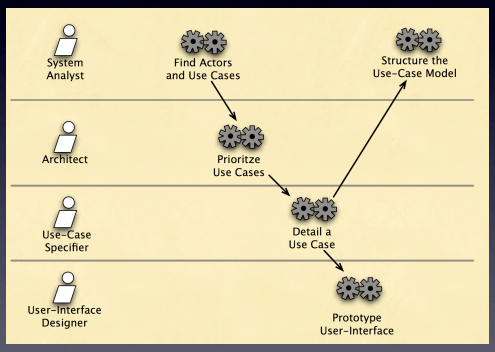
- During an iteration in any of the four phases you could do…
  - Requirements
  - Analysis
  - Design
  - Implementation
  - Test
- Obviously some of these will be more prominent in some phases than in others.
  - Implementation is more common in the Construction phase.

Core workflows

- The activities in a phase are called the core workflows.
  - Requirements (gathering), analysis, design, implementation, testing.
- The core workflows are described in terms of
  - Artefacts
    - The things produced. This could be, for example, an analysis model, a deployment model or test model.
  - Workers
    - The people/roles allocated to different tasks
  - Workflow
    - The activities that the workers follow to create the artefacts.

Sample workflow

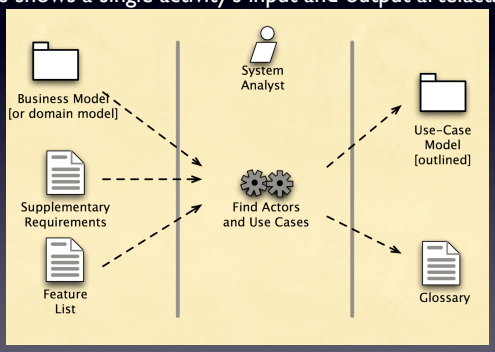
- The workflow models are activity diagrams, with different icons.



From a drawing in The Unified Software Development Process
Jacobson, Booch, Rumbaugh, Addison Wesley, 1999

A single activity

- This shows a single activity's input and output artefacts



From a drawing in The Unified Software Development Process
Jacobson, Booch, Rumbaugh, Addison Wesley, 1999

RUP details

- Each core workflow is described in terms of Artefacts, Workers and Workflows.
- For the Implementation core workflow they are
  - Artefacts
    - Implementation Model (subsystems, classes, interfaces associations)
    - Components (files, tables, documents, executables etc).
    - Implementation subsystem (Java packages, VB project etc).
    - Interface (like a high level Java interface for a subsystem, API)
    - Architecture Description
    - Integration Build Plan (how the system will be built)

RUP details

- Workers
  - Architect
  - Component Engineer (Programmer)
  - Systems Integrator
- Workflow
  - Architectural Implementation
  - Integrate System
  - Implement a Subsystem
  - Implement a Class
  - Perform Unit Test

Summary

- Reliably developing software requires a well understood process.
- Developing software reliably requires a well defined process.
- The waterfall model usually fails because feedback comes too late in the process.
- The Rational Unified Process is based on use cases, is iterative and incremental, allowing for early feedback.

Process overview

- Agile processes all emphasise...
  - Iterative development
  - Development based on user functions (stories, use cases)
  - Short time frames between releases (increments)

Extreme Programming

- Developed as a complete model of system development, including consideration of the people in the process
- Consists of
  - 5 values
  - 14 principles
  - 24 practices

www.agilexp.org/downloads/TheNewXP.pdf
Extreme Programming Explained: Embrace Change, Second Edition, Kent Beck

XP - 5 Values

- Communication
  - Between people - maximised, documents - easy to read
- Simplicity
  - Do the simplest thing that could possibly work
- Feedback
  - You should be able to measure how close to completion
- Courage
  - Action despite being afraid of consequences/failure
- Respect
  - Respect for others a human beings

XP - 14 Principles

Humanity	Economics
Mutual Benefit	Self-Similarity
Improvement	Diversity
Reflection	Flow
Opportunity	Redundancy
Failure	Quality
Baby Steps	Accepted Responsibility

XP - 24 Practices

Stories	Sit Together	Real Customer Involvement
Weekly Cycle	Whole Team	Incremental Deployment
Quarterly Cycle	Informative Workspace	Negotiated Scope Contract
Slack	Energized Work	Pay-Per-Use
Ten Minute Build	Pair Programming	Code and Tests
Continuous Integration	Team Continuity	Shared Code
Test-First Programming	Shrinking Teams	Single Code Base
Incremental Design	Root Cause Analysis	Daily Deployment

The environment

Sit Together
Whole Team
Informative Workspace
Energized Work
Pair Programming

- The first 3 are about the environment and how people interact.
- Sit together - be physically close
- Whole Team - People should accept that they are part of a team, and they all have to work together.
- Informative Workplace - should be easy to find out what is going on. Whiteboards, Stick notes etc.

Energized work

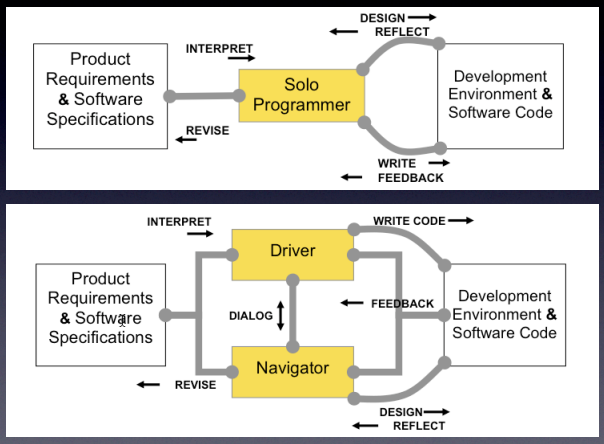
Sit Together
Whole Team
Informative Workspace
Energized Work
Pair Programming

- Energized work
  - People can work ok for 40 hours a week, for a short time at 50 hours a week.
  - More than that and your brain turns to...

Pair programming

- Navigator - Focuses on high level thinking
  - Thinks about how the module fits in with the program
- Driver - Focuses on low level details
  - Is in charge of the keyboard
- Pairs should form on an ad-hoc basis.

Pair programming



Pair Programming: When and Why it Works. Chong, Plummer, Leifer, Klemmer, Eris, Toye

Stories

Stories
Weekly Cycle
Quarterly Cycle
Slack
Ten Minute Build
Continuous Integration
Test-First Programming
Incremental Design

- Like use cases/requirements
- Free to discuss them with the client
- Perform estimation on stories - likely return vs. cost of implementation.
- Get customer involvement in choice of stories to implement.

Test first programming

- Stories
- Weekly Cycle
- Quarterly Cycle
- Slack
- Ten Minute Build
- Continuous Integration
- Test-First Programming
- Incremental Design

- Develop the tests first
- As the code hasn't been written, the tests should fail
- Work on making all the tests pass
- Stop coding when all the tests pass
- Prevents you from wandering into a programming limbo when you add things that aren't needed

Slack

- Stories
- Weekly Cycle
- Quarterly Cycle
- Slack
- Ten Minute Build
- Continuous Integration
- Test-First Programming
- Incremental Design

- Google allow people to have 1 day off a week to pursue their own goals
- XP suggests extra time in the schedule to allow for bad forecasting
- Build it in as a safety factor for yourself?

Code issues

- Code and Tests
- Shared Code
- Single Code Base
- Daily Deployment

- Code and Tests - don't maintain any other artefacts!
- Shared code - no one person owns the code. All are responsible for it.
- Single code base - no branches, forks
- Daily Deployment

Example

- A development team finds a bug in software that has been delivered.
- Which of the practices should come into effect to solve this problem?

XP through the RUP lens

- Are RUP and XP incompatible?
- What parts are similar?
- What parts are the same?

Contrasting the models

- XP eschews excessive documentation
- RUP says create appropriate documentation
- XP says release weekly
- RUP says release increments often
- XP says do pair programming
- RUP is silent on this issue
- XP says only keep code & tests
- RUP says maintain lots of artefacts

Process + Tools

- The Lockheed C-130J is particularly interesting because it was developed using a formal "correctness by construction" process using the SPARK Ada-based toolset from Praxis High Integrity Systems. The experience with that process was that, compared to industry norms for developing safety-critical avionics software, the C-130J development had a 10 times lower error rate, four times greater productivity, half as expensive a development process, and four times productivity increase in a subsequent project thanks to substantial reuse.

http://www.computerworld.com.au/index.php/id;447175928;pp;3

Process + Tools
Space shuttle software

- Four identical machines, running identical software, pull information from thousands of sensors, make hundreds of milli-second decisions, vote on every decision, check with each other 250 times a second. A fifth computer, with different software, stands by to take control should the other four malfunction.
- Consider these stats : the last three versions of the program -- each 420,000 lines long-had just one error each. The last 11 versions of this software had a total of 17 errors. Commercial programs of equivalent complexity would have 5,000 errors.

http://www.fastcompany.com/magazine/06/writestuff.html